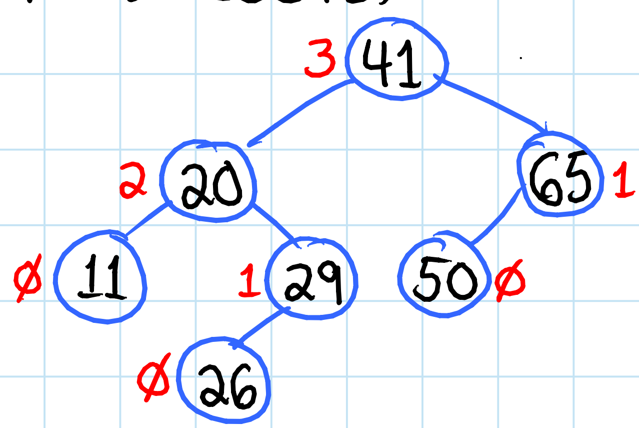
6.006	Lecture 6	Sept. 27, 2011

TODAY: Balanced BSTs

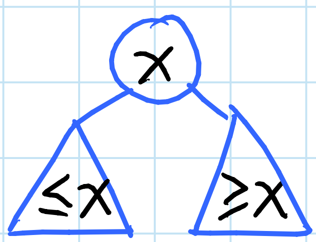
- The importance of being balanced
- AVL trees
  - definition & balance
  - rotations
  - insert
- Other balanced trees
- Data structures in general
- Lower bounds

Recall: Binary Search Trees (BSTs)

- rooted binary tree
- each node has
  - key
  - left pointer
  - right pointer
  - parent pointer



- BST property:

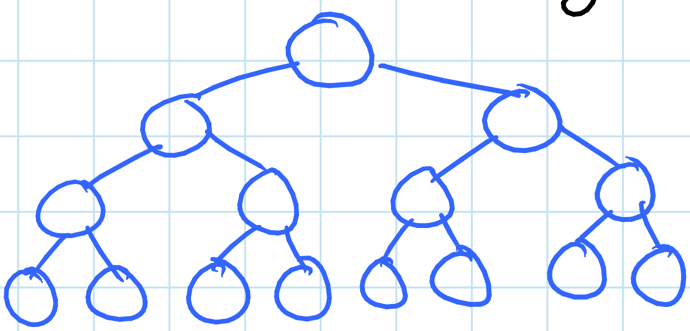


CLRS B.5

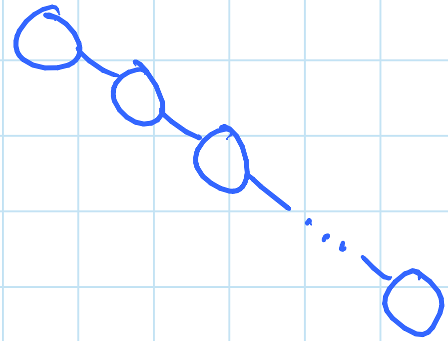
- height of node = length (# edges) of longest downward path to a leaf

The importance of being balanced:

- BSTs support insert, delete, min, max, next-larger, next-smaller, etc. in $O(h)$ time, where h = height of tree (= height of root)
- h is between $\lg n$ and n:



vs.



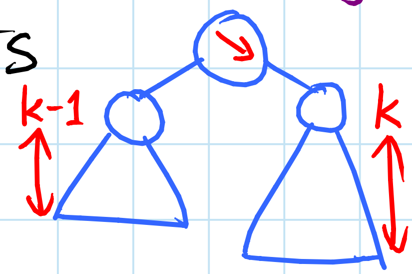
perfectly balanced

path

- balanced BST maintains $h = O(\lg n)$
  $\Rightarrow$ all operations run in $O(\lg n)$ time

AVL trees: [Adel'son-Vel'skii & Landis 1962]

for every node, require heights of left & right children to differ by at most ± 1



- treat nil tree as height -1
- each node stores its height (DATA STRUCTURE AUGMENTATION) (like subtree size)
(alternatively, can just store difference in heights)

Balance: worst when every node differs by 1

- let N_h = (min.) # nodes in height-h AVL tree

$\Rightarrow N_h = N_{h-1} + N_{h-2} + 1$
$\quad > 2 N_{h-2}$

$\Rightarrow N_h > 2^{h/2}$

$\Rightarrow h < 2 \lg N_h$

Alternatively: $N_h > F_h$ (nth Fibonacci number)

- in fact $N_h = F_{n+2} - 1$ (simple induction)
- $F_h = \varphi^h / \sqrt{5}$ rounded to nearest integer
  where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ (golden ratio)

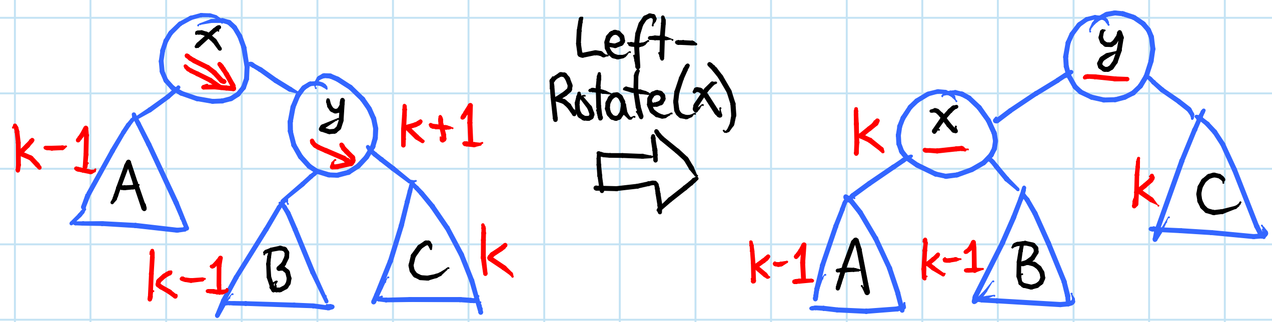
$\Rightarrow$ max. $h \approx \log_\varphi n \approx 1.440 \lg n$

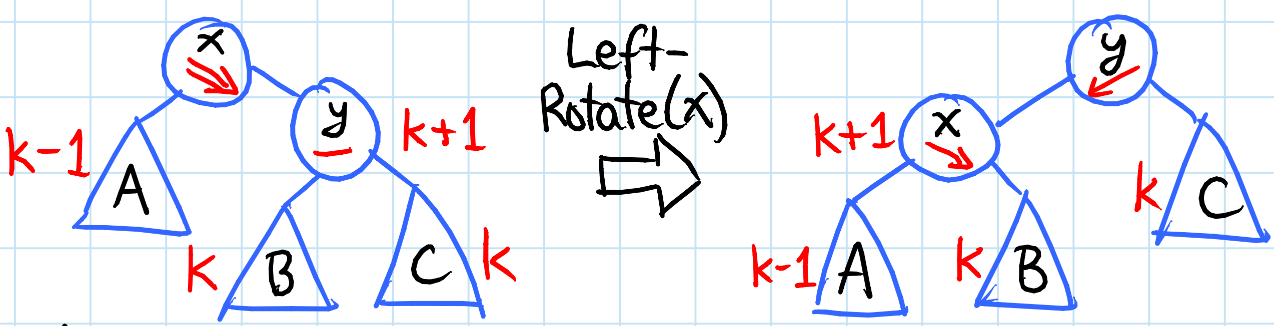
AVL insert:

1. insert as in simple BST
2. work your way up tree, restoring AVL property (and updating heights as you go)

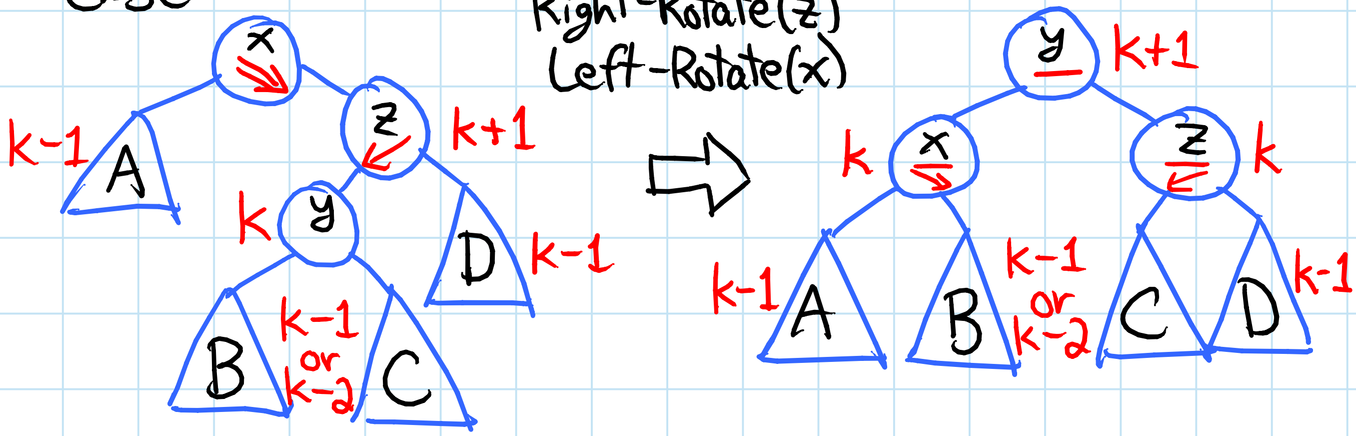
Each step:

- suppose x is lowest node violating AVL
- assume x is right-heavy (left case symmetric)
- if x's right child is right-heavy or balanced:





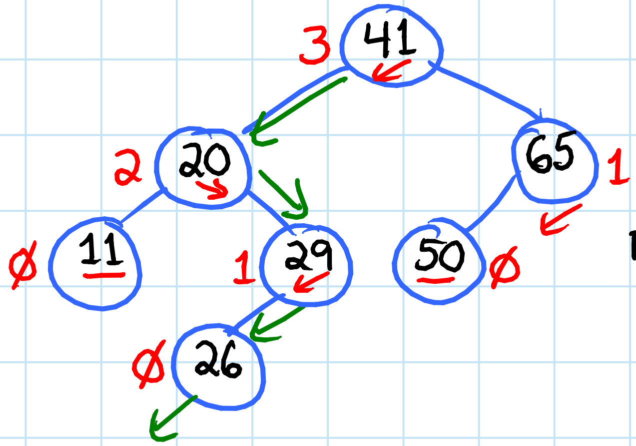
- else:



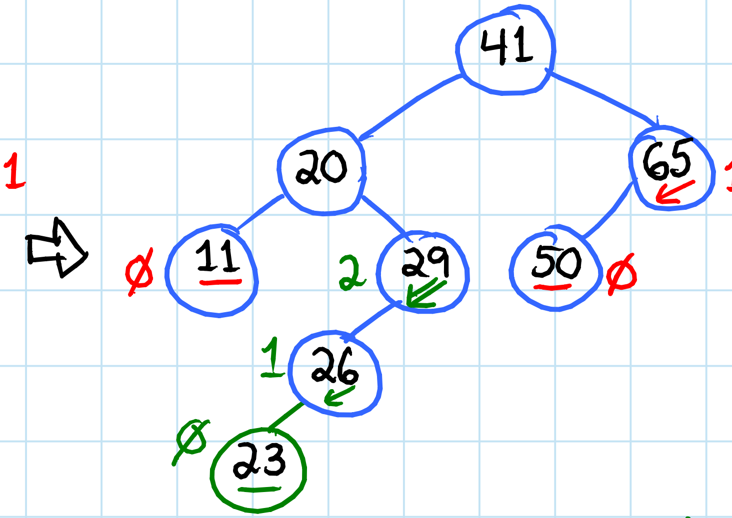
- then continue up to x's grandparent, greatgp, ...

Example:

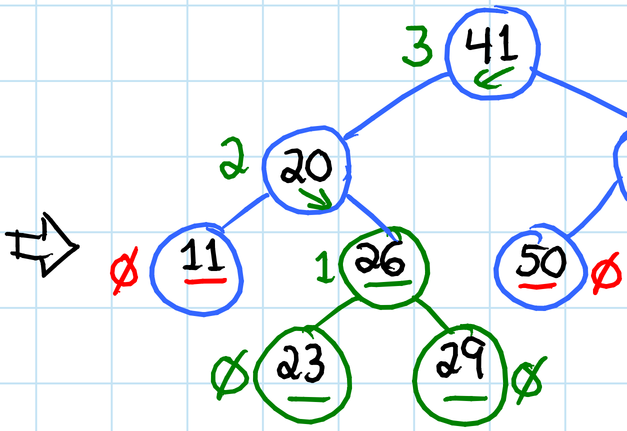
Insert(23)



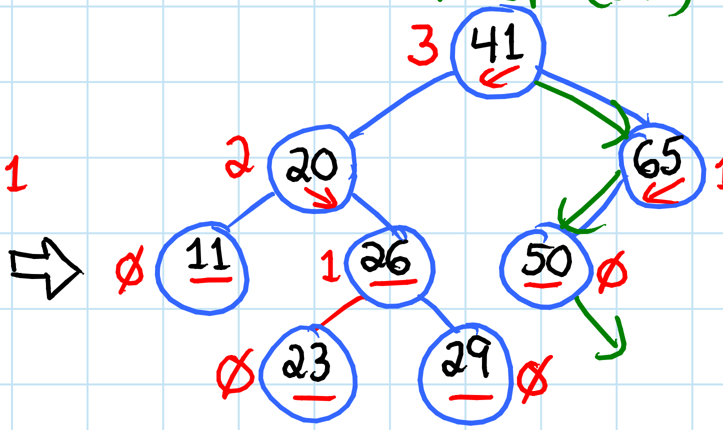
x=29: left-left case



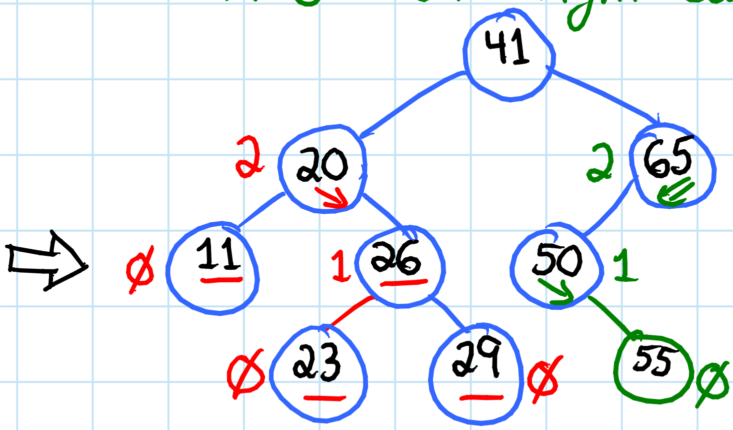
Done.



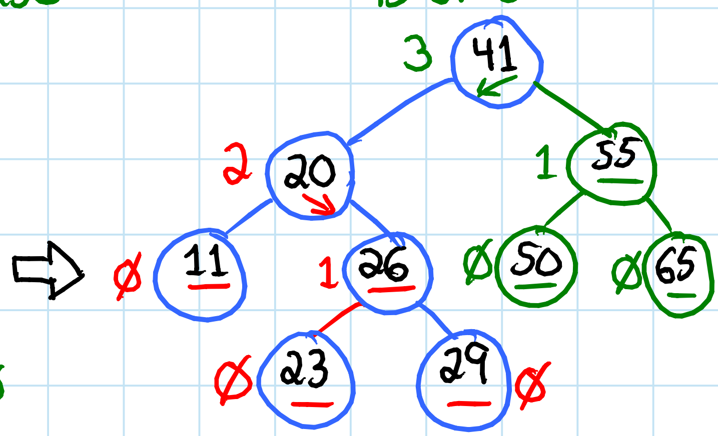
Insert(55)



x=65: left-right case



Done.



- in general may need several rotations before done with an Insert
- Delete(-min) is similar

AVL sort:

- insert each item into AVL tree $\Theta(n \lg n)$
- in-order traversal $\Theta(n)$

$\Theta(n \lg n)$

Balanced search trees: there are many!

- AVL trees [Adel'son-Velsii & Landis 1962]
- B-trees / 2-3-4 trees [Bayer & McCreight 1972] $\boxed{\text{CLRS } 18}$
- BB[α] trees [Nievergelt & Reingold 1973]
- red-black trees [CLRS ch. 13]
- A – splay trees [Sleator & Tarjan 1985]
- R – skip lists [Pugh 1989]
- A – scapegoat trees [Galperin & Rivest 1993]
- R – treaps [Seidel & Aragon 1996]

R = use random numbers to make decisions fast with high probability

A = "amortized": adding up costs for several operations $\Rightarrow$ fast on average

e.g. splay trees are a current research topic
- see 6.854 (Advanced Algorithms) & 6.851 (Advanced Data Structures)

Big picture:

Abstract Data Type (ADT): interface spec.
vs. Data Structure (DS): algorithm for each op.

- many possible DSs for one ADT
  e.g. much later, "heap" priority queue

Priority Queue ADT:

	heap	AVL tree
- Q = new-empty-queue()	$\Theta(1)$	$\Theta(1)$
- Q.insert(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
- x = Q.deletemin()	$\Theta(\lg n)$	$\Theta(\lg n)$
- x = Q.findmin()	$\Theta(1)$	$\Theta(\lg n)$ ↳ $\Theta(1)$

Predecessor/Successor ADT:

	heap	AVL tree
- S = new-empty()	$\Theta(1)$	$\Theta(1)$
- S.insert(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
- S.delete(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
- y = S.predecessor(x) ↳ next-smaller	$\Theta(n)$	$\Theta(\lg n)$
- y = S.successor(x) ↳ next-larger	$\Theta(n)$	$\Theta(\lg n)$